IoT Weather station with real-time data

1. Introduction

1.1. Background, limitations of traditional technologies, the need for real-time weather monitoring methods

Through the years, human beings have endeavoured to comprehend their surroundings. They have devised numerous tools to measure a variety of parameters. Those tools, typically the thermometer, barometer, and pyrometer, were created to gauge temperature, atmospheric pressure, and solar radiation, respectively[1]. However, these traditional instruments were limited to local use and were hard to see remotely.

Lately, various institutions have taken the initiative to establish their weather stations to gather weather-related data. These entities include airports, environmental agencies, and farming organizations. These weather stations are of a professional nature and their cost is determined by the precision and complexity of the sensors employed. For instance, the WMR200, manufactured by Oregon Scientific, is a weather station that exceeds a price tag of $1,000, rendering it unaffordable for small businesses.[2]. The high cost can be attributed to the utilization of proprietary software and hardware components. As a result, its installation and maintenance cost is significantly high. Besides the cost involved, most professional weather stations lack customization options, meaning that additional sensors cannot be added or removed. These stations are sold as closed products, and the generated data is typically stored locally within the station which poses the limitation of lack of flexibility. Presently, weather stations are predominantly found in major urban centers, airports, and climate research institutions. However, a drawback of this arrangement is that these stations are often located miles away from the areas they monitor. Accordingly, the collected data may not accurately reflect the actual weather conditions at the specific location, as it is influenced by the distance factor. Those variables such as temperature, rainfall index, and wind speed are highly susceptible to fluctuations, which are further exacerbated by distance variations. All of those listed drawbacks for the weather station could be a motivation for creating our local real-time weather station which will be implemented in this project.

1.2. Purpose and significance of the project, ambition

The advent of high-speed Internet and IoT has revolutionized data interconnectivity, extending to electronic devices. This project builds upon this connectivity, aiming to enhance local weather predictions through sophisticated sensor integration and advanced data processing techniques. The ability to predict weather conditions with higher accuracy has significant implications for individual uses. The ambition of this project is to amalgamate resolution data and localized sources to capture nuanced weather patterns pertinent to personal decision-making contexts.

1.3. Overview of the IoT weather station and its objectives

In the face of increasingly unpredictable climatic conditions, this paper details the design and implementation of an IoT-based local weather station with real-time prediction capabilities. Leveraging Arduino as a central controller and integrating various environmental sensors, the system consolidates data from external APIs with a pre-trained machine-learning model. This approach offers precise, real-time weather predictions, enhancing localized data for personal use in areas such as Oslo, Norway. The application developed is not limited to broad agricultural purposes but is tailored for individualized weather predictions to aid in daily decision-making, such as attire selection.

2. Solution Design

2.1 Design approach taken

The project introduces a weather monitoring system leveraging Arduino boards, aimed at creating a cost-effective station for storing climate data on a remote database server, accessible from any device, anywhere. The Arduino microcontroller serves as the central hub, coordinating various devices and sensors, including humidity and temperature sensors, to ensure smooth system operation. Once connected to the server, data from strategically placed sensors is promptly transmitted to the Arduino Cloud platform's web server via the integrated Wi-Fi module in Arduino Uno R4, facilitating data upload and preservation in the cloud.

The Arduino microcontroller plays a pivotal role in processing data captured from diverse sensors, which undergo preprocessing and signal conditioning to ensure compatibility for further analysis. Through the Arduino Cloud website, real-time monitoring and management

of the system become feasible, providing timely updates on changes in carbon monoxide (CO) levels, humidity, and temperature at the monitoring location. This invaluable data is stored in the cloud for easy access and continuous monitoring. Regular measurements of temperature, humidity, and CO content enable comprehensive data recording and analysis, and enhance situational awareness.

Moreover, leveraging the Arduino Cloud and a mobile app database which serves as a frontend to the Arduino for storing and sharing weather data facilitates real-time data tracking and visualization, integrating the weather station with modern technology. The proposed architecture emphasizes scalability, allowing for the addition of new sensors, such as rain sensors, to enhance surveillance capabilities. Designed with a layered architecture approach, the system aims to reduce complexity and costs while accommodating future modifications.

In terms of machine learning model development, the Edge Impulse platform is intended for weather prediction in this project. It simplifies the process of building and deploying machine learning models on edge devices like Arduino, enabling real-time data analysis directly on the edge without relying on a constant connection to the cloud.

2.2 Architecture of the system, including hardware and software components.

2.2.1 Sensors

The BME680 environmental compact sensor consists of a MEMS (Micro-Electro-Mechanical Systems) sensor for temperature, humidity, and gas resistance, along with an integrated ASIC (Application-Specific Integrated Circuit) for signal processing and communication. The BME680 sensor has a high sampling rate, typically around 1 sample per second. Although the BME680 sensor is highly proficient in delivering precise measurements of temperature, humidity, pressure, and gas resistance, it is crucial to acknowledge that its design emphasizes accuracy rather than speed. Consequently, it may not be the most suitable choice for applications that demand swift data acquisition, such as real-time weather monitoring systems. This constraint has the potential to impede the sensor's efficacy in projects where prompt data updates are imperative for timely decision-making or analysis.

The second feature of the BME680 sensor is that it communicates with a microcontroller (such as Arduino) using an I2C (Inter-Integrated Circuit) communication protocol. This allows the microcontroller to read data from the sensor and control its operation.

2.2.1.1 Air pressure sensor

This sensor measures the absolute pressure surrounding it, which varies according to weather and altitude. This sensor can be used to monitor weather changes, height measurements, or other tasks that require accurate pressure readings. [3]

2.2.1.2 Humidity and temperature

A humidity and temperature sensor is a sensor that can measure both the relative humidity and temperature of the surrounding environment. It is commonly used in various applications such as HVAC systems, weather monitoring stations, and indoor environmental quality monitoring systems.[3] The most common type of humidity and temperature sensor is the capacitive humidity sensor, which utilizes a thin polymer sheet to absorb or release water vapour in response to changes in relative humidity. This sensor can also measure temperature simultaneously. There are other forms of humidity sensors, such as resistive and thermal conductivity sensors, although they are less frequently used. Humidity and temperature sensors are available in different form factors and can communicate with other devices.

2.2.1.3 Gas resistance

The gas sensor can detect a broad range of gasses like volatile organic compounds (VOC)

2.2.2 Microcontroller:

The Arduino Uno R4 WiFi is an iteration of the classic Arduino Uno board with built-in WiFi capability. There are 2 components included which are an ATmega 4809 microcontroller with AVR architecture base and wifi module. The microcontroller runs the Arduino sketch uploaded to the board and handles all the I/O operations while the integrated WiFi allows the Arduino board to connect to wireless networks, enabling it to communicate with other devices and access the internet.

2.2.3 Arduino Cloud :

The platform provides some useful functionalities for this project such as visualization for real-time data, and remote control of IoT devices from anywhere with an internet connection. This includes sending commands, updating firmware, and configuring device settings without physically connecting it to a computer or re-uploading the code manually through OTA

(Over-The-Air) updates. In this project, The Arduino IDE software tool is deployed. The Arduino Integrated Development Environment (IDE) is a versatile application that is compatible across different platforms. It is coded using functions from C/C++ and is primarily used for writing and uploading codes to Arduino-compatible boards like Arduino Uno, Arduino Mega, Nodemcu, and others. The Arduino IDE converts the executable code into a text file encoded in hexadecimal format, which is then loaded into the Arduino board using a loader program in the board's firmware.

2.2.4 Cable Connection

To establish an electrical connection between components and facilitate the flow of electricity, a cable connection can be utilized. Most electrical connectors are categorized by gender, such as male and female components. For our purposes, female-to-female jumper wires are utilized, allowing for both ends to be plugged in. [4]

2.4. Details on how data collection, transmission, and processing are handled.

The project can be divided into four distinct stages: data collection, data storage in a database, analysis of the collected data, and prediction. In the first stage, weather data is collected by interfacing all available sensors with the Arduino boards. This allows for the recording of data from these sensors. To accomplish this, various in-built packages and libraries provided by the Arduino platform that are readily accessible are deployed. Additionally, the sensors can be calibrated based on their performance and the parameters that they provide. The values obtained from these sensors are then acquired and sent to the Arduino cloud for further processing. This processing involves tasks such as setting up the time interval for data upload to the cloud which in this case for every second, detecting any anomalies in the received data, monitoring changes in the weather data, and ensuring the proper functioning of all sensors.

3. Achieved Solution and Its Performance

3.1 Functionality and features of the system

3.1.1 Performance

The proposed system has been successfully implemented in hardware and the results have been displayed using the Arduino cloud platform. This system can be monitored through both

PCs and smartphone applications. Arduino Cloud has provided a suitable environment for analyzing and comparing sensor data through graphical representations.

These graphs from the Arduino Cloud interface provide valuable insights into temperature, humidity, air quality index based on air pressure, as well as C02 levels. Regularly updated, the temperature, humidity, AQI, and C02 levels are all presented at specific time intervals such as 7 days, 1 day, 1 hour, or live data for every 5 seconds so it could be easier for users to track.

Based on the obtained results, the proposed system can be considered a suitable Weather Monitoring System for domestic purposes, especially for personal uses such as choosing proper clothes.

3.1.2 Data transmission

Observations on the data logger indicate that data transmission is successful as long as the WiFi module which is integrated in Arduino Uno R4 is properly connected to the microcontroller and has internet access.

3.2 Insights into the performance of the system in real-life conditions, interpretation of data

A small manual test is conducted to verify the data accuracy produced by the weather station. This involves comparing the weather parameters measured by the station with data provided by the website of the Norwegian Meteorological Institute (MET Norway). The parameters checked include temperature, humidity, and air pressure.

Humidity

From using the BME 680 sensor, humidity measurements ranged from 41.27% to 64.46%. The discrepancy in humidity measurements between the BME 680 sensor and the data extracted from the official Norwegian Meteorological Institute was 7% at 16.53 PM on 26/4 , with an average error of 8%, compared to data from BME 680 sensor.

The highest air humidity recorded by the BME 680 sensor was 64.46% at 04:36 AM, and the lowest was 41.27% at 15:33 AM on the day 25/4/2024. This could be explained by thermal dynamics in Norway specifically at Kolbotn where the sensors data has been measured. The humidity levels follow a unique pattern, where mornings typically have higher humidity

compared to the afternoons. This occurrence is a result of a complex interaction between atmospheric dynamics and environmental conditions. During the night, as temperatures decrease, the air cools down, causing moisture to saturate and dew to form on surfaces. The calm morning atmosphere, coupled with lighter winds, allows this moisture to persist, leading to elevated humidity levels. Moreover, the presence of cloud cover overnight intensifies this effect by trapping moisture close to the ground. As the day advances and temperatures increase, the air's ability to retain moisture grows, resulting in a gradual decrease in humidity levels by the afternoon.

Air pressure

On 26/4 from 01:16 AM to 9:55 AM, air pressure data which shows a significant increase may indicate the approach or presence of high-pressure weather systems in the region of Kolbotn. High-pressure systems are typically associated with fair weather conditions, including clear skies and calm winds. Afterwards, a slight fall from 10:00 AM to 13:24 PM then keeps constant til 17.24 suggests a minor fluctuation in atmospheric conditions, which could be influenced by localized weather phenomena or changes in wind patterns. The highest air pressure recorded by the BMR 680 sensor was 998.2433 hPa at 20.30:48 PM, and the lowest was 996.3 hPa at 03:46 AM. The error in air pressure measurements between the BMP 180 sensor and the met.no was 9 hPa at around 17 PM, with an average error of 10 hPa.

Temperature

From the BME 680 sensor data, the highest recorded temperature was 13.98°C at 14:20 PM and the lowest was 7.32°C at 03:38 AM. Temperature data shows fluctuations which could be due to the varying duration of sunlight exposure and the angle of sunlight incidence. Smaller angles in the morning and evening, with the largest during midday, contribute to these fluctuations, compounded by varying weather conditions like sunny or cloudy skies.

Using the website of the Norwegian Meteorological Institute, the discrepancy in temperature measurements between the BME680 sensor and the met.no was a 3.7°C error at 17:00 PM, with an average error of around 3°C.

It is significantly important to note that all these tests utilize data from the meteorological institute at Blindern, which is situated at a higher altitude. Therefore, variations in the data

are anticipated because the real-time data from sensors is locally at Kolbon where it has been set up.

4. Highlights of Interesting Sub-Problems Solved and Not Solved

4.1 Challenging or interesting problems encountered during the project

4.1.1 Unstable WiFi Connections:

The project encountered difficulties with unstable WiFi connections, resulting in intermittent data transmission loss. Factors such as signal interference, network congestion, or limitations in the WiFi module's range or reliability may have contributed to this instability.

4.1.2 Integration of Machine Learning Module

The integration of a machine learning module proved challenging due to constraints on the chosen platform and project setup. Utilizing Edge Impulse, an API-based machine learning platform, required the Arduino board to be connected to a computer for data transfer, posing logistical issues as the IoT device was stationed outdoors. As a workaround, weather data was collected using a CSV approach for future machine learning model development.

4.1.3 Communication with Web Application

Establishing communication between the Arduino board and a web application with a single get endpoint on Google encountered difficulties, particularly with the board's HTTPS client capabilities. Despite efforts to upload SSL certificates as per Arduino's documentation, the HTTP client was unable to send requests to the web application, indicating a compatibility issue.

4.2. Solutions or alternative solutions

4.2.1 Addressing Unstable WiFi Connections

To tackle the problem of unreliable WiFi connections, error handling mechanisms were devised to automatically try to reconnect to WiFi when a loss of connection was detected. This ensured that data transmission would continue seamlessly even in the face of intermittent network disruptions. On a technical level, the error handling mechanism consisted of implementing regular checks of the WiFi connection status. If a disconnection

was identified, the system would initiate a reconnection attempt. This process typically included:

- Regular checks of the WiFi connection status at set intervals using built-in functions or libraries provided by the Arduino platform.
- If a disconnection was detected, the system would activate a reconnection attempt using appropriate WiFi reconnection functions or methods.
- Visual feedback on the WiFi connection status could be given to the user, such as through LED indicators or display messages, to indicate whether the device was connected to the network.
- To avoid continuous reconnection attempts in the case of persistent network problems, a retry mechanism with a configurable retry limit and timeout period could be put in place.

Furthermore, optimizations like adjusting the placement of the WiFi antenna, choosing less congested WiFi channels, or upgrading to more reliable WiFi modules with better signal strength could help alleviate the issue of unstable connections.

4.2.2 Exploring Alternative Communication Protocols

To overcome this obstacle, alternative strategies like utilizing a different communication protocol or exploring alternative hardware with improved HTTPS client capabilities could be considered for future versions of the project.

5. Suggestion for further works

5.1 Remaining limitations or areas for improvement

The IoT weather station currently relies on Arduino Cloud for data management and transmission. However, it faces limitations in scalability and functionality when compared to more advanced platforms such as Raspberry Pi. Likewise, the user interface for displaying real-time weather data may not be user-friendly. It is imperative to address these challenges to enhance the overall performance and usefulness of the IoT weather station.

5.2 What could be done?

It is important to note that every concept, technique, or endeavour has the potential for improvement. Our approach can still be enhanced by integrating it with various software types, incorporating additional sensors and hardware, and improving accessibility by creating more user-friendly interfaces.

The future scope of our project includes the following possibilities:

A. Utilizing different sensors to gather information about the environment, such as light intensity, pressure, wind speed, and direction. Expanding the range of sensor capabilities to encompass factors such as light intensity, pressure, wind speed, and direction significantly enhances the accuracy of machine-learning models. By incorporating this wider dataset, we can make more precise predictions regarding natural occurrences like earthquakes, thereby improving early warning systems and disaster preparedness initiatives.

B. Powering the weather station with solar panels, which serve as an excellent alternative energy source. Solar energy enhances the potential of portable weather stations by obviating the necessity for electric charging.

C. Attaching the instrument to a balloon instead of placement on the balcony as current practice, enabling data collection from higher altitudes and facilitating research on meteorological conditions in inaccessible regions.

D. The visualization provided by Arduino Cloud should be changed to make it user-friendly and informative simultaneously. Those changes could be the inclusion of some factors such as:

D.1 Color-coded categories

Assign different colours to temperature ranges, humidity levels, and air pressure.

D.2 Threshold alerts

Display a warning symbol or notification when potentially hazardous weather situations such as sudden high heat waves occur. In addition, the implementation of triggers on Arduino Cloud provides an opportunity for improved performance. By setting up triggers to activate based on specific conditions, such as temperature thresholds, individuals can automate the process of receiving notifications. For example, a trigger can be programmed to alert users

through the IoT remote application on their mobile device whenever the temperature surpasses or drops below a predetermined value.

In conclusion, the proposed system has demonstrated its potential for improving the current monitoring system. By exploring the prospects outlined above, further advancements can be made to enhance its functionality and applicability in various domains.

E. Dynamic Threshold Algorithm for Weather Interpretation and Personalized Recommendations

The visualization itself should function as a convenient tool for interpreting data. It may be beneficial to develop an algorithm that automates the process of establishing thresholds based on sensor data and other variables like temperature and humidity. By examining historical data and patterns, the algorithm can dynamically adjust these thresholds to accurately identify corresponding values. For example, when monitoring environmental conditions, the algorithm could set thresholds for temperature and humidity levels that indicate specific weather patterns or events. If the temperature surpasses a particular threshold, the algorithm could suggest wearing lightweight and breathable clothing to ensure comfort throughout the day. By utilizing real-time data and personalized preferences, the IoT device enables users to make well-informed decisions about their attire, thereby enhancing their overall comfort and well-being in different weather conditions. Furthermore, by considering the baseline between dry and rainy conditions, we can determine the intensity of rainfall. This allows for swift assessments, informed decision-making, and proactive responses in sectors such as agriculture, urban planning, and disaster management, to name a few.

6. References

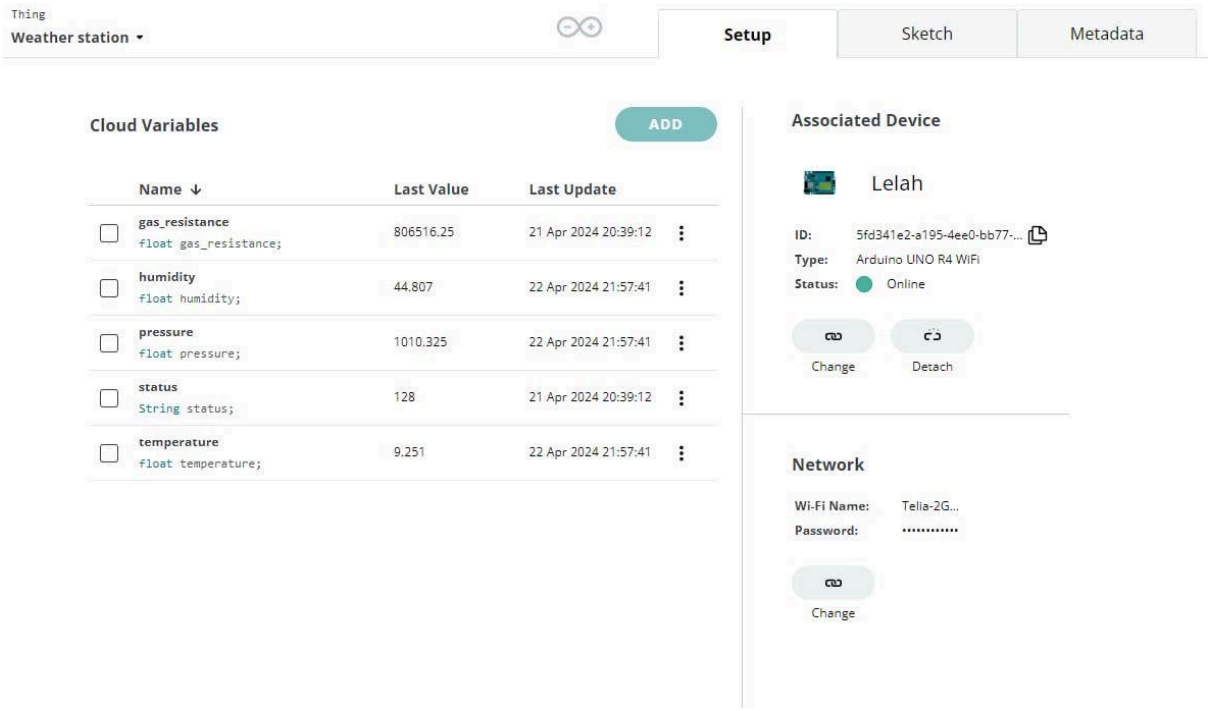
1. Tandan, N., Modi, R., Raut, A., Kanse, A., Karnam, H., Bedade, S., Uparkar, O., Mehta, S., "Design and Implementation of 10T Based Local Weather Station — An Experimental Setup," Department of Electronics and Telecommunication Engineering, Vidyalankar Institute of Technology, Mumbai, India.
2. Kusriyanto, M., "Weather Station Design Using 10T Platform Based on Arduino Mega," Electrical Engineering Department, Industrial Technology Faculty, Indonesia Islamic University, Yogyakarta, Indonesia.

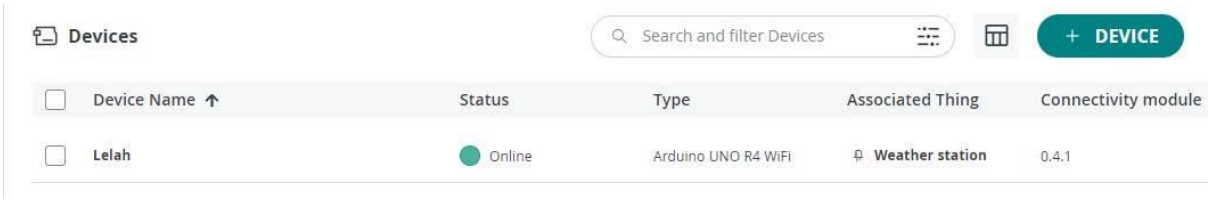
3. Sakar I, V., Kumar, S. N., Aasrith, A. S., Karthick, G. S. A., Raghuraman, L., & Balaji, S., "Comprehensive Study of Weather Prediction Using 10T and Machine Learning," School of Electrical Engineering, Vellore Institute of Technology, India.
4. Kapoor, P., "Cloud Based Weather Station Using IoT Devices," Dept of CSE, IIT Guwahati, Assam, India
5. Chakraborty, S., Joshi, T., Agarwal, S., "Internet of Things (IOT) Based Cost-Effective Weather Monitoring Station," Department of Electrical and Electronics Engineering, Rajiv Gandhi Institute of Petroleum Technology, Jais, India
6. Bin Shahadat, A. S., Ayon, S. I., Khatun, M. R., "Efficient 10T Based Weather Station," Dept. of Computer Science and Engineering, Khulna University of Engineering & Technology, Khulna, Bangladesh
7. Singh, D. K., Jerath, H., Raja, P., "Low Cost 10T Enabled Weather Station," Embedded Systems, SEEE, Lovely Professional University, Phagwra, Punjab, India
8. Pandey, J., Raghav, A., Bisariya, S., "Implementation of an IOT Live Weather Monitoring System," Computer Science and Engineering, Bipin Tripathi Kumaon Institute of Technology, Dwarahat, Uttarakhand, India. Electronics and Communication, ABES Engineering College, Ghaziabad, India
9. Arbain @ Sulaiman, I. b., Sadli, M. D. D., "An IoT-based Smart Garden with Weather Station System," Faculty of Electrical Engineering (Computer Engineering), Universiti Teknologi Mara (UiTM), Shah Alam, Selangor.
10. Varghese, L., Deepak, G., Santhanavijayan, A., "A 10T Analytics Approach for Weather Forecasting using Raspberry Pi 3 Model," Department of Computer Science and Engineering, National Institute of Technology Tiruchirappalli, Tamil Nadu, India
11. Byabazaire, J., O'Hare, G. M. P., Collier, R., Delaney, D., "Dynamic Data Source Selection: A Case of Weather Stations for 10T Applications," * School of Computer Science, University College Dublin, Dublin, Ireland. t School of Electrical and Electronic Engineering, University College Dublin, Dublin, Ireland. School of Computer Science and Statistics, Trinity College Dublin, Dublin, Ireland
12. Marwal, C., Othman, S. B., Sakli, H., "10T Based Low-cost Weather Station and Monitoring System for Smart Agriculture," IMACS Research Laboratory, National Engineering School of Gabes, Gabes University, Gabes, Tunisia. PRINCE Laboratory Research, ISITcom, Hammam Sousse, University of Sousse, Tunisia. EITA Consulting, 5 Rue du Chant des oiseaux, Montesson, France.

7 Appendix

Code snippets

1. *Retrieving data from sensors*





Weather_station.ino | ReadMe.adoc | thingProperties.h | Secret Tab

```
/*
  Sketch generated by the Arduino IoT Cloud Thing "Weather station"
  https://create.arduino.cc/cloud/things/0cd22dff-978f-4472-b4f9-026fed660f55

  Arduino IoT Cloud Variables description

  The following variables are automatically generated and updated when changes are made to the Thing

  String status;
  float gas_resistance;
  float humidity;
  float pressure;
  float temperature;

  Variables which are marked as READ/WRITE in the Cloud Thing will also have functions
  which are called when their values are changed from the Dashboard.
  These functions are generated with the Thing and added at the end of this sketch.
*/

#include "ArduinoGraphics.h"
#include "Arduino_LED_Matrix.h"
#include "bme68xLibrary.h"
#include "thingProperties.h"


#ifndef PIN_CS
#define PIN_CS SS
#endif

Bme68x bme;
ArduinoLEDMatrix matrix;

WiFiSSLClient client;
const char host[] = "script.googleusercontent.com/";
const int port = 443;
const String WEATHER_STATION_SENSOR_DATA_ID = "AKfycbyLNAoKVdD046mgVTwKwpc7p1rS--i4LDvsFHJk1Iuex0YJgrPW0t44kG26lPqxSsxLTw";


byte happy[8][12] = {
    {0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,1,1,1,0,0,1,1,1,0,0},
    {0,0,1,1,1,0,0,1,1,1,0,0},
    {1,0,0,0,0,0,0,0,0,0,0,1},
    {1,1,0,0,0,0,0,0,0,0,1,1},
    {0,1,1,1,1,1,1,1,1,1,1,0},
    {0,0,1,1,1,1,1,1,1,1,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0}
};

byte sad[8][12] = {
    {0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,1,1,1,0,0,1,1,1,0,0},
    {0,0,1,1,1,0,0,1,1,1,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,1,1,1,1,1,1,1,1,0,0},
    {0,1,1,1,1,1,1,1,1,1,1,0},
```

```
byte sad[8][12] = {
    {0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,1,1,1,0,0,1,1,1,0,0},
    {0,0,1,1,1,0,0,1,1,1,0,0},
    {0,0,0,0,0,0,0,0,0,0,0,0},
    {0,0,1,1,1,1,1,1,1,1,0,0},
    {0,1,1,1,1,1,1,1,1,1,1,0},
    {1,1,0,0,0,0,0,0,0,0,1,1},
    {1,0,0,0,0,0,0,0,0,0,0,1}
};

// initialsize the onboard led matrix
void initializeLeds() {
  Serial.begin(57600);
  matrix.begin();
}

// initialise the bme680 sensor
void initializeSensor() {
  SPI.begin();
  Serial.begin(115200);

  while (!Serial)
    delay(10);

  /* initializes the sensor based on SPI library */
  bme.begin(PIN_CS, SPI);

  if(bme.checkStatus()) {
    if (bme.checkStatus() == BME68X_ERROR) {
      Serial.println("Sensor error:" + bme.statusString());
      return;
    }
    else if (bme.checkStatus() == BME68X_WARNING) {
      Serial.println("Sensor Warning:" + bme.statusString());
    }
  }

  /* Set the default configuration for temperature, pressure and humidity */
  bme.setTPH();

  /* Set the heater configuration to 300 deg C for 100ms for Forced mode */
  //bme.setHeaterProf(300, 100);
}

void initializeCloud() {
  // Initialize serial and wait for port to open:
  Serial.begin(9600);
  // This delay gives the chance to wait for a Serial Monitor without blocking if none is found
  delay(1500);

  // Defined in thingProperties.h
  initProperties();
```

```
void initializeCloud() {
  // Initialize serial and wait for port to open:
  Serial.begin(9600);
  // This delay gives the chance to wait for a Serial Monitor without blocking if none is found
  delay(1500);

  // Defined in thingProperties.h
  initProperties();

  // Connect to Arduino IoT Cloud
  ArduinoCloud.begin(ArduinoIoTPreferredConnection);

  /*
     The following function allows you to obtain more information
     related to the state of network and IoT Cloud connection and errors
     the higher number the more granular information you'll get.
     The default is 0 (only errors).
     Maximum is 4
 */
  setDebugMessageLevel(2);
  ArduinoCloud.printDebugInfo();
}

void updateCloud() {
  ArduinoCloud.update();

  bme68xData data;

  bme.setOpMode(BME68X_FORCED_MODE);
  delayMicroseconds(bme.getMeasDur());

  // fetches data and update the variables in arduino cloud
  if (bme.fetchData())
  {
    bme.getData(data);
    temperature = float(data.temperature);
    // convert pressure from pascal to hectopascal (hPa)
    pressure = float(data.pressure)/100;
    humidity = float(data.humidity);
    gas_resistance = float(data.gas_resistance);
    status = String(data.status);

    Serial.print(String(temperature) + ", ");
    Serial.print(String(pressure) + ", ");
    Serial.print(String(humidity) + ", ");
    Serial.print(String(data.gas_resistance) + ", ");
    Serial.println(status);
  };
}

/* checks the leds and update the onboard led matrix accordingly.
  Displays a happy face if connected to wifi, otherwise a sadface.
*/
void updateLeds() {
  if (WiFi.status() == WL_CONNECTED) {
```

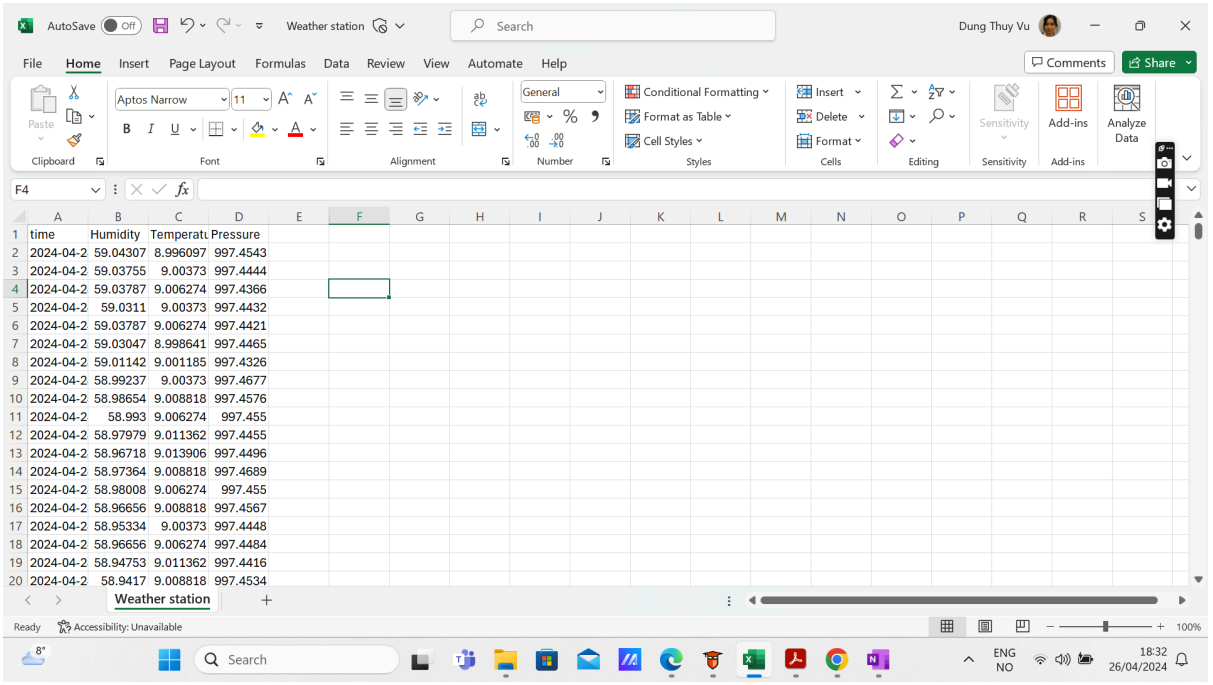
```
void updateLeds() {
  if (WiFi.status() == WL_CONNECTED) {
    matrix.renderBitmap(happy, 8, 12);
  } else {
    matrix.renderBitmap(sad, 8, 12);
    Serial.println("Wifi status: Lost wifi connection, reconnecting");
    // Attempts to reconnect to wifi if connection was lost
    WiFi.begin(SSID, PASS);
  }
}

void sendData(String temperature, String humidity, String pressure) {
   if (client.connect(host, 443)) {
     String url = "/macros/s/" + WEATHER_STATION_SENSOR_DATA_ID + "/exec?temperature=" + temperature + "&humidity=" + humidity + "&pressure=" + pressure;
     client.println("GET " + url + " HTTP/1.1");
     client.println("Host: script.google.com");
     client.println("User-Agent: ArduinoWeatherStation");
     client.println("Connection: close");
   } else {
     Serial.println("Failed to connect to server");
   }
}

void setup() {
  initializeLeds();
  initializeSensor();
  initializeCloud();
}

void loop() {
  Serial.println("Wifi status: " + String(WiFi.status()));
  updateCloud();
  updateLeds();
  //sendData(String(temperature), String(humidity), String(pressure));
}
```

Weather_station.ino | ReadMe.adoc | thingProperties.h | Secret Tab

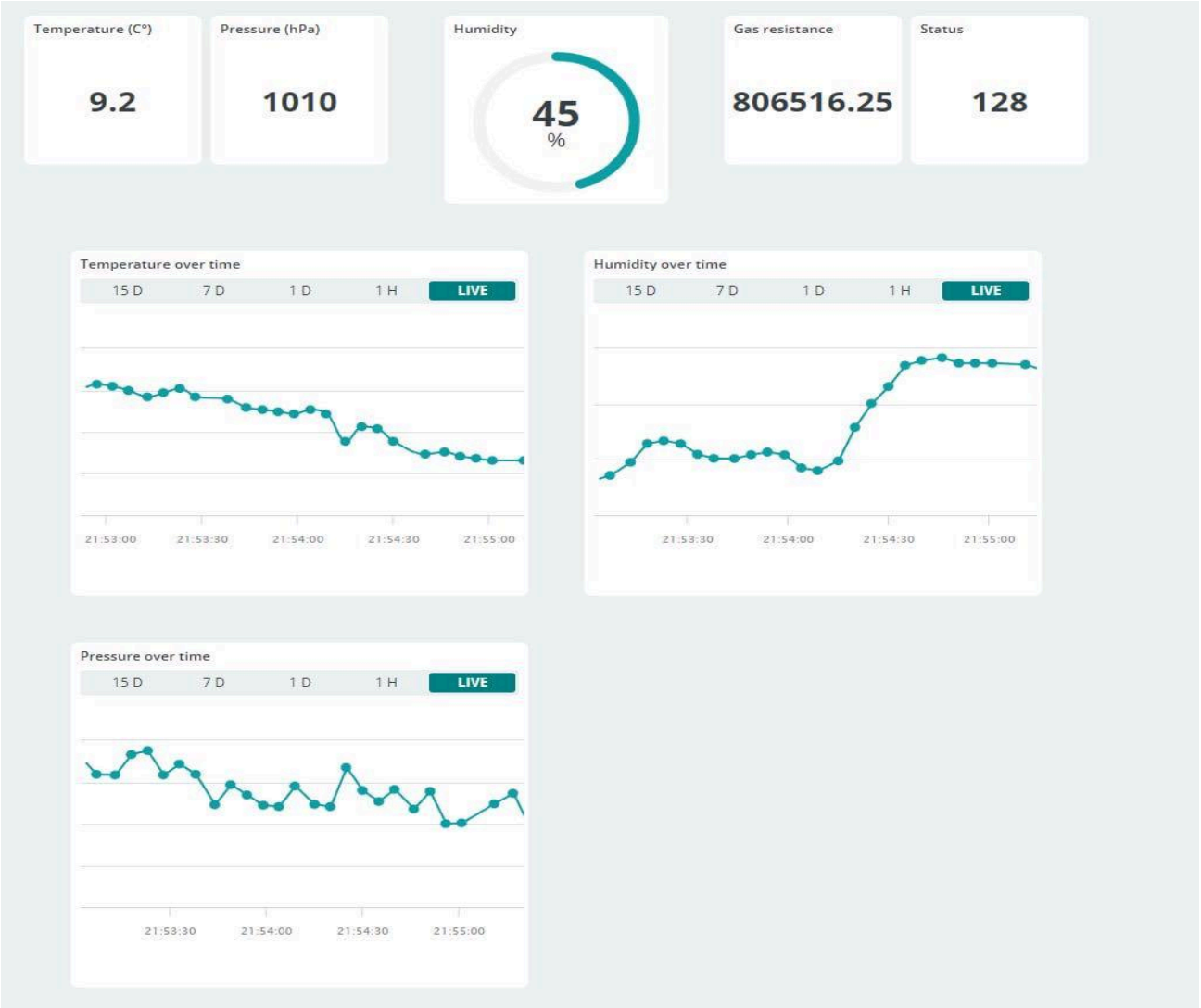
```
// Code generated by Arduino IoT Cloud, DO NOT EDIT.

#include <ArduinoIoTCloud.h>
#include <Arduino_ConnectionHandler.h>

const char SSID[]     = SECRET_SSID;    // Network SSID (name)
const char PASS[]     = SECRET_OPTIONAL_PASS;    // Network password (use for WPA, or use as key for WEP)


String status;
float gas_resistance;
float humidity;
float pressure;
float temperature;

void initProperties(){

  ArduinoCloud.addProperty(status, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(gas_resistance, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(humidity, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(pressure, READ, ON_CHANGE, NULL);
  ArduinoCloud.addProperty(temperature, READ, ON_CHANGE, NULL);

}

WiFiConnectionHandler ArduinoIoTPreferredConnection(SSID, PASS);

```

2. *Web application to display data on Google Spreadsheet and csv file of sensor data.*

```
// app url: https://script.google.com/macros/s/AKfycbyLNAoKVdD046mgVTwKwpc7p1rS--i4LDvsFHJk1Iuex0YJgrPW0t44kG26lPqxSsxLTw/exec
// sheet: https://docs.google.com/spreadsheets/d/12CMphUrPV9f1yyTa1E-g6kpzG9IwyXd5E0v0ojORd1w/edit#gid=0

function doGet(e) {
  Logger.log( JSON.stringify(e) );
  var result = 'Ok';
  if (e.parameter == 'undefined') {
    result = 'No Parameters';
  }
  else {
    var sheet_id = '12CMphUrPV9f1yyTa1E-g6kpzG9IwyXd5E0v0ojORd1w';
    var sheet = SpreadsheetApp.openById(sheet_id).getActiveSheet();
    var newRow = sheet.getLastRow() + 1;
    var rowData = [];
    var timestampt = new Date().toISOString()
    rowData[0] = timestampt;
    for (var param in e.parameter) {
      Logger.log('In for loop, param=' + param);
      var value = stripQuotes(e.parameter[param]);
      Logger.log(param + ':' + e.parameter[param]);
      switch (param) {
        case 'temperature': //Parameter
          rowData[1] = value; //Value in column B
          result = 'Written on column B';
          break;
        case 'humidity': //Parameter
          rowData[2] = value; //Value in column C
          result += ' ,Written on column C';
          break;
        case 'pressure':
          rowData[3] = value; //Value in column C
          result += ' ,Written on column D';
          break;
        default:
          result = "unsupported parameter";
      }
    }
    Logger.log(JSON.stringify(rowData));
    // Write new row below
    var newRange = sheet.getRange(newRow, 1, 1, rowData.length);
    newRange.setValues([rowData]);
  }
  // Return result of operation
  return ContentService.createTextOutput(result);
}


/**
* Remove leading and trailing single or double quotes
*/
function stripQuotes( value ) {
  return value.replace(/^["']|['"]$/g, "");
}

```

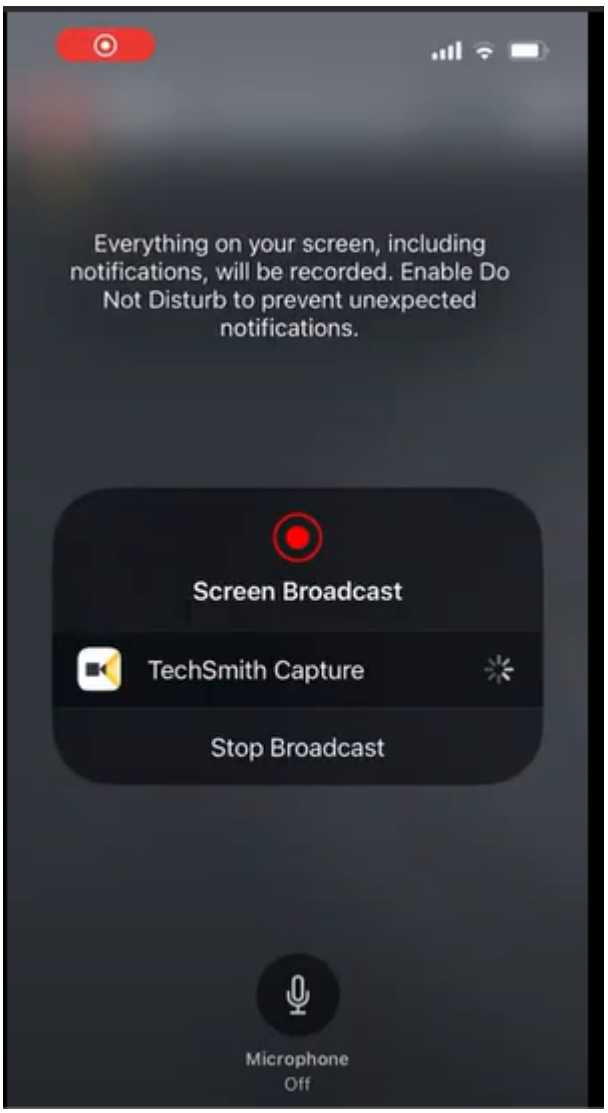

AutoSave Off
Weather station
Search
Dung Thuy Vu
File Home Insert Page Layout Formulas Data Review View Automate Help
Comments
Share
Aptos Narrow
11
General
Conditional Formatting
Format as Table
Cell Styles
Insert
Delete
Format
Sensitivity
Add-ins
Analyze Data
Clipboard
Font
Alignment
Number
Styles
Cells
Editing
Sensitivity
Add-ins
F4
time Humidity Temperatu Pressure
2024-04-2 59.04307 8.996097 997.4543
2024-04-2 59.03755 9.00373 997.4444
2024-04-2 59.03787 9.006274 997.4366
2024-04-2 59.0311 9.00373 997.4432
2024-04-2 59.03787 9.006274 997.4421
2024-04-2 59.03047 8.998641 997.4465
2024-04-2 59.01142 9.001185 997.4326
2024-04-2 58.99237 9.00373 997.4677
2024-04-2 58.98654 9.008818 997.4576
2024-04-2 58.993 9.006274 997.455
2024-04-2 58.97979 9.011362 997.4455
2024-04-2 58.96718 9.013906 997.4496
2024-04-2 58.97364 9.008818 997.4689
2024-04-2 58.98008 9.006274 997.455
2024-04-2 58.96656 9.008818 997.4567
2024-04-2 58.95334 9.00373 997.4448
2024-04-2 58.96656 9.006274 997.4484
2024-04-2 58.94753 9.011362 997.4416
2024-04-2 58.9417 9.008818 997.4534
Weather station
Ready
Accessibility: Unavailable
100%
Search
ENG NO
18:32 26/04/2024

Interface of IoT Weather Station



Video about mobile app's interface for IoT weather station

439638766_7203324219765645_8161822322279313929_n.mp4


Everything on your screen, including notifications, will be recorded. Enable Do Not Disturb to prevent unexpected notifications.
Screen Broadcast
TechSmith Capture
Stop Broadcast
Microphone
Off